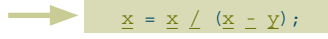


möglichen Fehler, deren Abwesenheit nachgewiesen werden muss, ist die Division durch Null in der hervorgehobenen Zeile. Das Problem lässt sich auch als Frage formulieren: Können x und y zur gleichen Zeit identische Werte annehmen?

```
int where_are_the_errors (int input)
{
    int x,y,k,l;
    k = input / 100;
    x = 2;
    y = k + 5;
    while (x < 10)
    {
        x++;
        y = y + 3;
    }

    if ((3*k + 100) > 43)
    {
        y++;
        x = x / (x - y);
    }

    return x;
}
```



Zur Beantwortung dieser Frage muss das Verifikationstool x und y statisch berechnen, und zwar Zeile für Zeile und von Anfang an. Ein rein visuelles Code-Review ließe sich auf diese Weise niemals mit ausreichender Gründlichkeit durchführen.

Um derartige numerische Probleme zu lösen, müssen Code-Verifikationstools die Daten möglichst genau verstehen und modellieren, beispielsweise durch Eingrenzung als komplexe Vielecke oder andere geometrische Formen (Abb. 2).

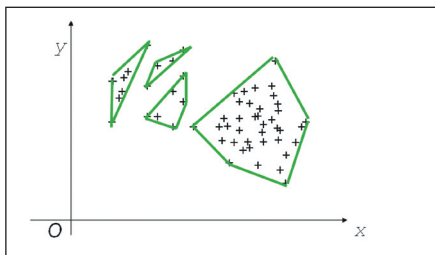


Abb. 2. Ansammlungen von Datenpunkten lassen sich durch komplexe Vielecke approximieren; dies wird intern von Verifikations-Tool genutzt.

Lesen von Pointern (Aliasing)

Wenn sich Daten auf andere Daten beziehen, was in C etwa durch Pointer realisiert wird, muss das Verifikationstool ursprüngliche und darauf bezug nehmende Daten erkennen. Anschließend muss es eine Pointer-Analyse durchführen, die ermittelt, auf welche Weise diese Daten sich gegenseitig beeinflussen.

Im folgenden Code muss das Verifikationstool erkennen, dass das Aliasing durch eine bedingte Neuweisung in der ersten hervorgehobenen Zeile verändert wird.

```
# define RANGE (X,MIN,MAX) \
X=random; while ((X<MIN) ||
(X>MAX)) X = random;
volatile int random;
int x,y;
int f(int *ptr)
{
    int results;
    if (random) ptr = &y;
    *ptr = *ptr + 1;
    results = x + y + *ptr;
    return results;
}
void main(void)
{
    int tmp;
    RANGE(x, 0, 10);
    RANGE(y, -10, 10);
    tmp = f(&x);
    tmp = (2*tmp*tmp + 3*tmp + 1) /
(tmp + 19);
    tmp++;
}
```



Abb. 3. Oben: In diesem Modell werden Variablen in zwei Schritten abgespeichert. Unten: Der aus dem Modell generierte Code zeigt, wie Wertebereiche von Daten auch aus Endlosschleifen heraus weitergegeben werden können.

Am Ende dieser Anweisung zeigt ptr entweder auf x oder auf y , abhängig davon, welcher Pfad ausgeführt wurde. In beiden Fällen ist tmp größer als -19 . In der zweiten hervorgehobenen Zeile kann also niemals eine Division durch Null auftreten.

Modellierung von Programmier-Konstrukten

Ein Verifikationstool muss die Syntax verstehen, die verschiedenen Programmier-Konstrukten wie For-Schleifen, If-Then-Else oder unbestimmten Bedingungen zu Grunde liegt. Es muss also beispielsweise mathematisch berechnen, was innerhalb einer Endlosschleife geschieht, ohne diese auszuführen (Abb. 3).

Code-Verifikation mit PolySpace

Delphi Diesel Systems entwickelt für OEM-Kunden aus dem Automobilsektor Diesel-Einspritztechnologien, die Motoren leiser, sparsamer, schadstoffärmer und durchzugsstärker machen. Delphi setzt PolySpace-Produkte ein, um Software-Module direkt nach deren Erstellung zu analysieren bevor Funktionstests fertiger Bauteile durchgeführt werden.

CSEE Transport, ein führender Entwickler von Signal- und Leitsystemen für Hochgeschwindigkeitszüge, nutzt PolySpace-Produkte zur Verifikation der Sicherheitssoftware seiner Signalsysteme. Mit klassischen Verifikationsmethoden lassen sich bestimmte Fehler im handgeschriebenen ADA-Code grundsätzlich nicht entdecken. CSEE Transport setzt PolySpace-Produkte außerdem zur Validierung einzelner Module vor deren Integration ein.

Das **NATO HAWK Management Office** (NHMO) unterhält verschiedene komplexe, missionskritische Anwendungen für das Boden-Luft-Raketensystem HAWK. Zur Einhaltung von Zuverlässigkeits-Standards muss das NHMO-Team Laufzeitfehler identifizieren und beseitigen. Es führt hierzu detaillierte Analysen zur Dynamik der Anwendungen mit PolySpace-Produkten durch, anhand derer Prioritäten für Code-Reviews festgelegt werden.

Die **Glucolight Corporation** hat ein auf Bildverarbeitung beruhendes, nicht-invasives System zur Dauerüberwachung des Blutzuckerspiegels entwickelt. Zur Sicherstellung der Zuverlässigkeit der von Zulieferern bereitgestellten Zur Verbesserung der Zuverlässigkeit der von Zulieferern bereitgestellten Embedded Software sowie zur Vorbereitung auf die FDA-Zertifizierung wurde neuer oder veränderter C++-Code klassenweise mit PolySpace-Produkten verifiziert. Hierbei konnten mit PolySpace fehleranfällige Strukturen identifiziert und in Folgeversionen des Codes ersetzt werden.

Lesen sprachspezifischer Konstruktionen

In bestimmten Sprachen, auch in C++, kann hinter einer einzigen Programmzeile eine große Menge von Rechenoperationen stecken. Durch objekt-orientierte Konstrukte wie Vererbung, Polymorphismen und Templates wird C++-Code außerdem schnell unübersichtlich. Das Verifikations-Tool muss mathematisch ermitteln, welche Anweisungen in jeder Programmzeile ausgeführt werden.

Umfassende Unterstützung anspruchsvollster Aufgaben

Der Beweis der Fehlerfreiheit von Quellcode für Laufzeitfehler wie Überläufe, Divisionen durch Null und Array-Zugriffen außerhalb des gültigen Bereichs ist eine sehr aufwändige Aufgabe, die formale mathematische Methoden und ein vollständiges Verständnis der Semantik einer Sprache voraussetzt. PolySpace-Produkte machen diese Aufgabe durchführbar, weil sie den Quellcode

verifizieren, ohne ihn zu kompilieren oder auszuführen. Eben dieses Arbeiten mit dem Quellcode garantiert die Anwendbarkeit der Methode auf handgeschriebenen, automatisch generierten und beliebig aus beiden Sorten gemischten Programmcode.

Neben der Code-Verifikation testen PolySpace-Tools auch auf MISRA-C®-Konformität. Für Anwendungen mit automatisch aus Simulink®-Modellen oder UML-Diagrammen generiertem Programmcode gibt es außerdem PolySpace Link-Produkte, die die Verifikationsergebnisse wieder mit dem Ursprungsmodell verknüpfen.

PolySpace-Produkte haben sich in allen Bereichen der Entwicklung und Verifikation von Software bewährt (siehe Kasten). So wurden sie etwa als Quality-Gate für von unterschiedlichen Entwicklerteams geschriebenen Code eingesetzt oder in die Submit- und Build-Phasen für separate Software-Komponenten integriert und konnten

so den Entwicklern ein schnelleres Feedback geben. Zusammen mit den Link-Produkten für Modellierungs-Tools wie Simulink und Rhapsody® wurde PolySpace außerdem in der Designphase eingesetzt, um Systemingenieuren oder Algorithmenentwicklern durch Analysen des automatisch generierten Codes beim Aufspüren von Schwächen eines Entwurfs zu helfen. ■

Quellen

POLYSPACE PRODUKTE

www.mathworks.de/nn8/polyspace

WHITE PAPER: Code-Verifikation und Identifikation von Laufzeitfehlern durch abstrakte Interpretation

www.mathworks.de/nn8/abstract_interpretation

